Semantic Data Markets: a Flexible Environment for Knowledge Management [*]

Roberto De Virgilio
Università Roma Tre, Italy
devirgilio@dia.uniroma3.it

Giorgio Orsi
University of Oxford, UK
giorgio.orsi@cs.ox.ac.uk

Letizia Tanca
Politecnico di Milano, Italy
tanca@elet.polimi.it

Riccardo Torlone
Università Roma Tre, Italy
torlone@dia.uniroma3.it

ABSTRACT

We present NYAYA, a system for the management of Semantic-Web data which couples a general-purpose and extensible storage mechanism with efficient ontology reasoning and querying capabilities. NYAYA processes large Semantic-Web datasets, expressed in multiple formalisms, by transforming them into a collection of *Semantic Data Kiosks*. NYAYA uniformly exposes the native meta-data of each kiosk using the Datalog$^{\pm}$ language, a powerful rule-based modelling language for ontological databases. The kiosks form a *Semantic Data Market* where the data in each kiosk can be uniformly accessed using conjunctive queries and where users can specify user-defined constraints over the data. NYAYA is easily extensible and robust to updates of both data and meta-data in the kiosk and can readily adapt to different logical organization of the persistent storage. The approach has been experimented using well-known benchmarks, and compared to state-of-the-art research prototypes and commercial systems.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Query Processing, Rule-based databases; H.2.3 [**Database Management**]: Query languages

General Terms

Management, Performance, Experimentation

Keywords

ontological databases, Datalog$^{\pm}$, query rewriting, query answering, semantic data management, semantic web

[*]This research has been partially funded by the European Commission, Programmes IDEAS-ERC, Projects no.227977-SMScom and no. 246858-DIADEM and from the Oxford Martin School's grant no. LC0910-019-Extreme and Ubiquitous Computing.

1. INTRODUCTION

Ever since Tim Berners Lee presented the design principles for Linked Data[1], the public availability of Semantic Web data has grown rapidly. Today, many organizations and practitioners are all contributing to the "Web of Data", building RDF repositories either from scratch or by publishing, in RDF, data stored in traditional formats. The adoption of ontology languages such as RDFS and OWL supports this trend by providing the means to semantically annotate Web data with meta-data, enabling ontological querying and reasoning. Despite the fact that storing, reasoning over, and querying large datasets of semantically annotated data in a flexible and efficient way represents a challenging area of research and a profitable opportunity for industry [16], semantic applications using RDF and linked-data repositories set performance and flexibility requirements that, in our opinion, have not yet been satisfied by the state of the art of data-management solutions. In particular, current semantic data-management systems present some common shortcomings: (i) they usually operate within a language-dependent framework, implementing the reasoning and query-processing algorithms for a specific ontology language; (ii) they hardly adapt to requirements of changing the underlying physical organization (e.g., for optimization purposes); (iii) to bring the expressiveness and performance desiderata to terms, most systems unnecessarily restrict the allowed combination of ontology-modeling constructs, although in many cases decidability and tractability of reasoning and query answering are syntactically identifiable on a per-ontology basis.

We discuss possible solutions to these problems by presenting NYAYA[2], an environment for Semantic-Web data management which provides – in the same order as above – (i) flexible and uniform ontology-reasoning and querying capabilities over semantic data sets expressed in different formalisms, (ii) an efficient and, most importantly, general and extensible storage policy for Semantic-Web datasets, which can adapt to different query engines, exploiting their optimization strategies, (iii) the possibility to syntactically check whether the meta-data in a given repository can be queried efficiently and to use the subset of the language that keeps the complexity of the process at the required level. The last aspect is particularly significant. Suppose for instance that the combined use of two language constructs could potentially lead to undecidability or to a complexity increment;

[1]`http://linkeddata.org/`
[2]Nyaya is the name of the school of logic in the Hindu philosophy.

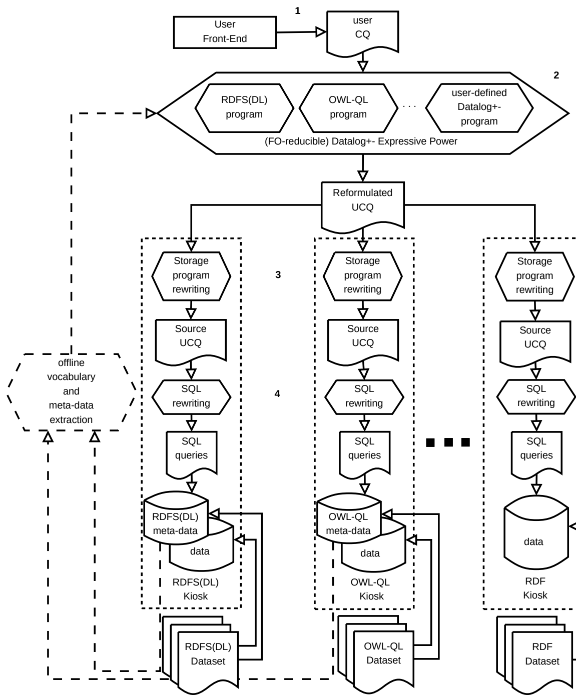


Figure 1: A Semantic Data Market

normally, with a Semantic Web language the two constructs would not appear in the language together, thus severely reducing expressiveness. Conversely, NYAYA does not restrict a-priori the language but checks each set of meta-data for decidability and tractability.

Reasoning and querying in NYAYA is based on Datalog$^{\pm}$ [5, 6], a family of rule-based languages that extends Datalog [8] to allow a controlled class of unsafe rules, i.e., rules with existentially quantified variables in their heads. Datalog$^{\pm}$ captures the most common ontology languages for which query answering is tractable, and provides efficiently-checkable, syntactic conditions for decidability and tractability. To our knowledge, this is the first full and generic implementation of a Datalog$^{\pm}$ engine.

NYAYA allows the construction of persistent repositories of Semantic-Web data that we call *Semantic Data Kiosks*. As shown at the bottom of Figure 1, a kiosk is populated by importing an available RDF dataset, possibly coupled with constraints defined on its content expressed in some Semantic-Web language such as RDF(S) and OWL (or variants thereof). In order to allow inferencing, native vocabularies and meta-data are extracted and represented as (first-order) Datalog$^{\pm}$ constraints. In addition, the entities of a kiosk are mapped to their actual representation in the storage by means of a set of non-recursive Datalog rules called *storage program*. This enables the separation of concerns between, on the one hand, the reasoning and query-processing algorithms and, on the other hand, the logical organization of the storage, which can be changed without affecting querying. A collection of kiosks constitutes what we call a *Semantic Data Market*, a place that exposes the content of all the kiosks in a uniform way and where users can issue queries and collect results, possibly by specifying additional constraints over the available data using Datalog$^{\pm}$. An important aspect of NYAYA is its flexibility. First, whenever a fresh Semantic-Web source is made available, a new kiosk can be easily built from it by extracting its meta-data and importing its data; in this way, its content is promptly available to the users of the semantic data market. Second, if a user wants to query the same kiosk by adopting a different set of constraints, the query is automatically reformulated accordingly and issued to the kiosk.

Summarizing, our main contributions are the following:

- the definition of the Semantic Data Kiosk, an extension of the standard Knowledge Base where the ontological constraints and the logical organization of persistent information are uniformly represented in the same language;
- the associated efficient and model-independent storage mechanism for Semantic-Web data, which guarantees the separation of concerns between the reasoning capabilities and the underlying organization of data;
- a powerful rule-based inference and query-answering engine over massive data sets based on Datalog$^{\pm}$, whose efficiency is confirmed by our experimental results over widely accepted benchmarks.

The rest of the paper is organized as follows. Section 2 presents the notion of semantic data kiosk and the adopted storage and querying techniques of NYAYA. Section 3 discusses related literature, while Section 4 illustrates the implementation of NYAYA and the experiments that allowed us to evaluate NYAYA's performance. Finally, Section 5 draws some conclusions and delineates our future work.

2. SEMANTIC DATA MARKETS

Semantic Data Kiosk. The basic building block of NYAYA has three components:

1. an *ontological program* Σ_O, made of a set of Datalog$^{\pm}$ [5, 6] rules over a collection of first-order predicates $\mathcal{O}$, called *ontology predicates*, that represent the ontological entities of the current application;
2. a *storage program* Σ_S, made of a set of full TGDs having, as head, a (single) atom with predicate in $\mathcal{O}$ and, as body, a conjunction of atoms over a set of first-order predicates $\mathcal{S}$, called *storage predicates*, that represent constructs of the metamodel $\mathcal{M}$;
3. a database D over $\mathcal{S}$.

A *Semantic Data Market* is just a collection of Semantic Data Kiosks.

Storage. Following an approach for the uniform management of heterogeneous data models [2], in NYAYA data and meta-data are extracted from the sources and represented using a *meta-model* $\mathcal{M}$ made of a generic set of *constructs*, each of which represents a primitive used in a concrete Semantic-Web language (such as RDF, RDFS and OWL). This has two main advantages: (i) it provides a framework in which different Semantic-Web languages can be handled in a uniform way and (ii) it allows the definition of *language-independent* reasoning capabilities. The constructs of $\mathcal{M}$ are chosen by factoring out common modeling primitives of different models. For instance, $\mathcal{M}$ includes the construct CLASS that is used in both RDFS and OWL to represent a set of resources having common characteristics. Each construct is associated with an identifier, a name, a set of *features*, and a set of references to other constructs. NYAYA relies on a relational implementation of the meta-model where each construct corresponds to a relational table. The approach is easily extensible; for instance, generalization hi-

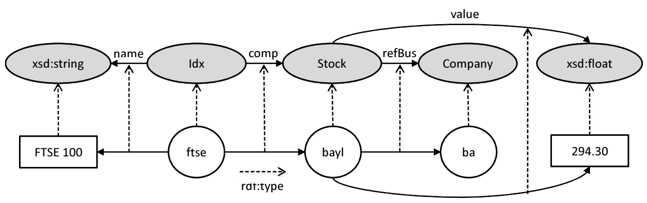
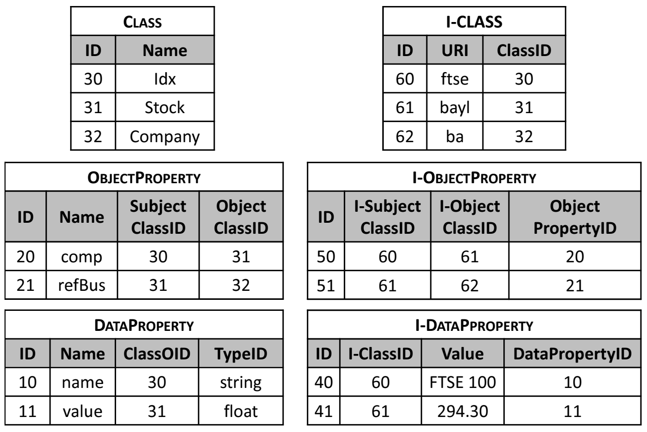


Figure 2: A running example.

erarchies can be added to $\mathcal{M}$ by specifying constructs that capture the notions of subclass and subproperty [2].

As an example, the RDF/RDFS stock-exchange scenario of Figure 2 is represented using $\mathcal{M}$ as illustrated in Figure 3 (where some fields are omitted for the sake of readability). The example involves three classes: `Stock`, `Company`, and



Class

ID	Name
30	Idx
31	Stock
32	Company

I-CLASS

ID	URI	ClassID
60	ftse	30
61	bayl	31
62	ba	32

ObjectProperty

ID	Name	Subject ClassID	Object ClassID
20	comp	30	31
21	refBus	31	32

I-ObjectProperty

ID	I-Subject ClassID	I-Object ClassID	Object PropertyID
50	60	61	20
51	61	62	21

DataProperty

ID	Name	ClassOID	TypeID
10	name	30	string
11	value	31	float

I-DataPproperty

ID	I-ClassID	Value	DataPropertyID
40	60	FTSE 100	10
41	61	294.30	11

Figure 3: Representation of the running example in the meta-model used for data storage.

`Idx`, which models financial indexes. Each class is represented by a tuple in the Class table. The instances of these classes, namely `ftse` (the financial index FTSE 100), `bayl` (the British Airways PLC stock) and `ba` (the British Airways company), are represented by tuples in the I-Class table. The relationships `comp` and `refBus`, modeling the stock composition of a financial index and the company related to each stock respectively, are represented by tuples in the ObjectProperty table. Similarly, the attribute `name` of a financial index and the attribute `value` of a stock are represented in the DataProperty table. Instances of these relationships are represented in the i-ObjectProperty and i-DataProperty tables, respectively. In particular, the `bayl` stock, which is a component of the financial index `ftse`, refers to the `ba` company and has value `294.30` pounds. Note that data and meta-data are managed in a uniform way. In Section 4 we illustrate the indexing and partitioning techniques adopted to guarantee good performance when the above tables become very large.

Ontological and storage programs. The top half of Table 1 shows a possible terminological knowledge over the stock-exchange scenario illustrated above. The first constraint specifies that each stock index must have a name, while the second one specifies that a stock has a value, a reference business (`refBus`), and it is part of a financial index. The third constraint specifies that a stock is a particular form of financial instrument (`FinIns`), and the following one defines a new class representing investments. The last

Table 1: Semantic Data Kiosk: Stock Exchange

	Datalog$^{\pm}$rules
$\Sigma_{\mathcal{O}}$	idx(X) $\rightarrow \exists Y$name(X,Y).
	stock(X) $\rightarrow \exists YZW$ value(X,Y), comp(X,Z), idx(Z), refBus(X,W), company(W).
	stock(X) $\rightarrow$ finIns(X).
	investment(X) $\rightarrow \exists YZW$ product(X,Y), stock(Y), buyer(X,Z), qty(X,W).
	company(X) $\rightarrow \exists Y$ market(X,Y).
	loan(X) $\rightarrow$ finIns(X).
	company(X), buyer(Y,X) $\rightarrow$ marketActor(X).
	comp(X,Y) $\rightarrow$ stock(X).
	comp(X,Y) $\rightarrow$ idx(Y).
	loan(X), stock(X) $\rightarrow \perp$. (NC)
	comp(X,Y) $\wedge$ comp(X,Z) $\rightarrow$ Y=Z. (EGD)
$\Sigma_{\mathcal{S}}$	idx(X) $\leftarrow$ I-CLASS(Z$_0$,X,Z$_1$), CLASS(Z$_1$,'Idx').
	comp(X,Y) $\leftarrow$ I-OBJECTPROPERTY(Z$_0$,Z$_1$,Z$_2$,Z$_3$), I-CLASS(Z$_1$,X,Z$_6$), I-CLASS(Z$_2$,Y,Z$_7$), OBJECTPROPERTY(Z$_3$,'comp',Z$_4$,Z$_5$).
	name(X,Y) $\leftarrow$ I-DATAPROPERTY(Z$_0$,Z$_1$,Y,Z$_2$), DATAPROPERTY(Z$_2$,'name',Z$_3$,Z$_4$), I-CLASS(Z$_1$,X,Z$_5$).

two constraints are: (i) a negative constraint specifying that loans and stocks are disjoint classes, and (ii) a functional constraint on the relationship `comp`. The bottom half of Table 1 provides three examples of storage-program rules for the class `idx`, the object property `comp` and the data property `value`, respectively. These rules describe how concepts, roles and attributes of the ontology are represented in the meta-model used for storage and are automatically derived when a new semantic data kiosk is built.

Querying. Let us now consider a semantic data kiosk $\mathcal{K} = (\Sigma_{\mathcal{O}}, \Sigma_{\mathcal{S}}, D)$. Querying K requires to consider both the constraints in $\Sigma_{\mathcal{O}}$ and in $\Sigma_{\mathcal{S}}$.

In Nyaya querying and reasoning are reduced to conjunctive query answering under *first-order reducible constraints* [17], i.e., given a query q, it is possible to construct a first-order query q_{rew} such that for every database D the answers to q over D given the constraints in $\Sigma_{\mathcal{O}} \cup \Sigma_{\mathcal{S}}$ are answers of q_{rew} over D (i.e., the query q_{rew} "embeds" the constraints in $\Sigma_{\mathcal{O}} \cup \Sigma_{\mathcal{S}}$). This class of constraints corresponds to the class of all the non-recursive Datalog programs for which data-complexity of query answering with respect to a fixed query and a fixed set of rules is in uniform AC^0 [15]. This allows Nyaya to delegate both reasoning and querying to the underlying relational database engine thus obtaining the same efficiency as for traditional database queries. Notice that the rules of $\Sigma_{\mathcal{S}}$ always constitute a safe and non-recursive Datalog program. Therefore, if $\Sigma_{\mathcal{O}}$ is FO-reducible then conjunctive-query answering over D under $\Sigma_{\mathcal{O}} \cup \Sigma_{\mathcal{S}}$ is also FO-reducible [10], implying that the organization of the data in the persistent storage does not affect the complexity of query answering. Based on this result, the construction of a new Semantic Data Kiosk from a fresh data source requires only to check for FO-reducibility of its meta-data. This check needs also to be applied whenever a user introduces a new user-defined Datalog$^{\pm}$ program to be used on top of an existing stored dataset.

Nyaya provides an implementation of two FO-reducible members of the Datalog$^{\pm}$ family: Linear Datalog$^{\pm}$ [5], and Sticky Datalog$^{\pm}$ [6]. Both languages are strictly more expressive than DL-*Lite* [7], the best-known language for tractable ontology-based data access and thus together offer a powerful formalism for expressing ontological constraints. Let Q be a user conjunctive query expressed in Datalog-like syntax over a semantic data kiosk. Following Figure 1, the processing of a query Q against a semantic data kiosk $\mathcal{K} = (\Sigma_{\mathcal{O}}, \Sigma_{\mathcal{S}}, D)$ proceeds as follows.

1. The CQ Q is reformulated using the rules in $\Sigma_{\mathcal{O}}$ by applying TGD-Rewrite [11], a backward-chaining resolution algorithm that produces a perfect rewriting $Q_{\mathcal{O}}$ of Q with respect to $\Sigma_{\mathcal{O}}$. By virtue of the FO-reducibility of Sticky and Linear Datalog$^{\pm}$, $Q_{\mathcal{O}}$ is a union of conjunctive queries.
2. Every $Q^i_{\mathcal{O}} \in Q_{\mathcal{O}}$ is rewritten in terms of the storage tables using the rules in $\Sigma_{\mathcal{S}}$. For each $Q^i_{\mathcal{O}}$, the result is another CQ $Q^i_{\mathcal{S}}$ that is perfect rewriting of $Q^i_{\mathcal{O}}$ with respect of $\Sigma_{\mathcal{S}}$. The union of all the $Q^i_{\mathcal{S}}$ forms a perfect rewriting $\hat{Q}$ of Q with respect to $\Sigma_{\mathcal{O}} \cup \Sigma_{\mathcal{S}}$.
3. The query $\hat{Q}$ is rewritten in SQL and executed over the underlying DBMS.

Consider the Semantic Data Kiosk in Table 1 and the query $Q: \ Q(A) \leftarrow \mathit{name}(C, A), \mathit{comp}(B, C), \mathit{stock}(B)$ retrieving the names of the financial indexes with at least one quoted stock (i.e., a sanity check query). After the application of the steps above, the final query is easily translated into the following SQL statement to be executed on the underlying relational database.

```
SELECT DP.Value
FROM I-OBJECTPROPERTY AS OP
     I-DATAPROPERTY as DP
WHERE DP.I-ClassID = OP.I-ObjectClassID AND
      OP.ObjectPropertyID = '20' AND
      DP.DataPropertyID = '10'
```

This dramatic simplification is due to the fact that the knowledge of the meta-model allows us to eliminate a high number of automatically generated subqueries, reducing the number of joins to be executed, with a noticeable improvement in performance.

3. RELATED WORK

Existing systems for the management of Semantic-Web data can be discussed according to two major issues: *storage* and *querying*.

Considering the storage, two main approaches can be identified: the first focuses on developing *native storage systems* (such as AllegroGraph[3] or OWLIM[4]) to exploit ad-hoc optimisations, while the second (e.g., Sesame[5], TAP[6], Jena[7], Virtuoso[8] and the semantic extensions implemented in Oracle Database 11g R2 [9]) make use of traditional DBMSs, *e.g.* relational and object-oriented. Generally speaking, native storage systems are more efficient in terms of load and update time, whereas the adoption of mature data management systems has the advantage of relying on consolidate and effective optimisations. Indeed, a drawback of native approaches consists in the need for re-thinking query-optimization and transaction-processing techniques. Differently from the other approaches, NYAYA provides a *general-purpose* storage policy for Semantic-Web data sets that allows the uniform management of different ontological formalisms and language-independent reasoning capabilities. A rule-based mapping between the ontological entities and their representation in the storage separates concerns between the query-processing algorithms and the physical organization of data, which can be changed without affecting querying. Our current relational implementation takes advantage of RDBMS optimization techniques based on indexing and partitioning.

On the querying side, if we assume the absence of ontological constraints, the efficiency of query processing depends only on the logical and physical organization of the data and on the query language complexity. However reasoning is a fundamental requirement of Semantic-Web applications, and efficient reasoning algorithms are now available for several ontology languages based on description logics. A first family of systems [4, 19, 18] materialises all the inferences; this dramatically improves query-processing performance, but loading time is inflated because all the complexity of reasoning is deferred to the loading time and could potentially generate an exponential blow-up of space occupation. Moreover, update management is also critical since the inferred data must be updated accordingly. Full materialisation is therefore suitable for stable data sets, especially when the query patterns are known in advance. A second class of systems [1, 14, 3, 12] including NYAYA executes inferencing on-the-fly through *intensional query reformulation*, thus the complexity of query processing depends on both the logical organisation of the data and the expressiveness of the adopted data language and of the queries. This approach has many advantages: in the first place, space occupation is reduced to the minimum necessary to store the data set and it is possible to define a suitable trade-off for inference materialisation depending on which queries are executed most frequently. Another advantage is the robustness to updates: since inferences are computed every time, there is no need for incremental updates. The major drawback of on-the-fly inferencing is the impact of reasoning time on query processing. In NYAYA, the separation between the ontological entities and the organization of data allow us to push the rewriting up to the level of the persistent storage, increasing the efficiency of query answering. In addition, the Datalog$^{\pm}$ language adopted in NYAYA allows to test decidability and tractability of query answering for a given ontology in polynomial time.

4. EXPERIMENTS

One of the most interesting contributions of NYAYA is its overall efficiency that makes it appealing for general-purpose applications. In this section we provide a detailed experimental evaluation that supports this claim.

NYAYA has been implemented in Java and its inference component has been built by extending the IRIS Datalog engine[9]. Oracle 11g R2 has been used for the persistent storage, exploiting the native referential partitioning technique of the DBMS. In particular, tables storing instance-level predicates are horizontally partitioned with respect to the foreign keys that refer to the tables storing schema-level predicates. A prototype implementation of NYAYA is available on the Web[10]. We compared NYAYA with two well-known representatives of semantic-data management systems that rely on full materialization: BIGOWLIM[11] and

[3]http://www.franz.com/products/allegrograph/
[4]http://www.ontotext.com/owlim/
[5]http://www.openrdf.org/
[6]http://tap.stanford.edu/.
[7]http://jena.sourceforge.net/
[8]http://virtuoso.openlinksw.com/

[9]http://www.iris-reasoner.org/
[10]http://pamir.dia.uniroma3.it:8080/nyaya/
[11]http://www.ontotext.com/owlim/big/

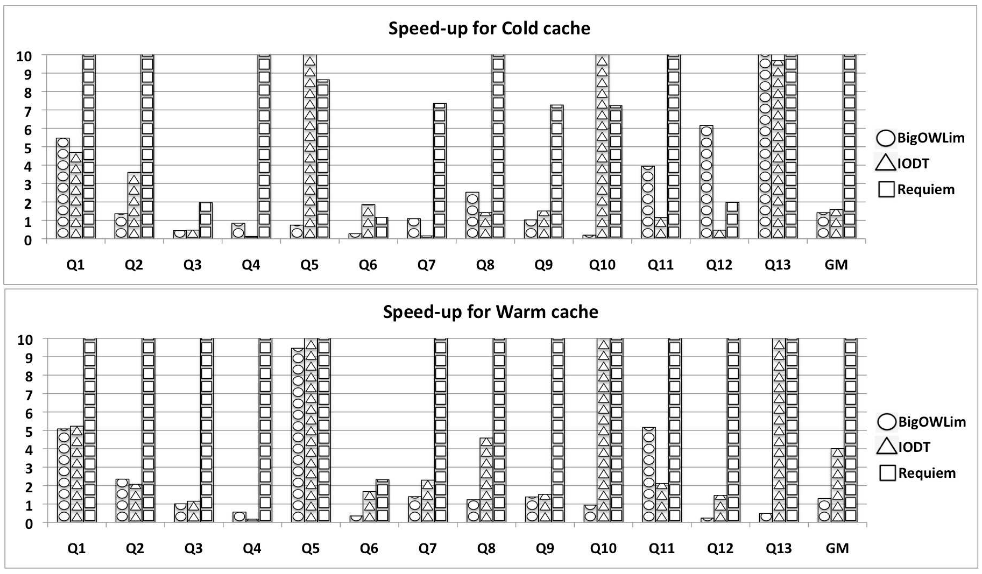


Figure 4: Performance ratio on UOBM50 between other semantic-data management systems and Nyaya

Table 2: Data Loading

	BigOWLim	IODT	Nyaya
LUBM(50)	8 (mins)	77 (mins)	9,18 (mins)
UOBM(50)	18 (hours)	65 (hours)	0,5 (hours)
Wikipedia3	71 (hours)	181 (hours)	2,38 (hours)

IODT[12]. In particular, we used BigOWLim v.3.3 over the File System, and IODT v.1.5 equipped with the Scalable Ontology Repository (SOR). In addition, we compared the rewriting technique of Nyaya with the on-the-fly inferencing mechanisms of REQUIEM [14] over the Nyaya storage meta-model, since REQUIEM does not rely on a specific back-end. All the experiments have been performed on a dual-quad core 2.66GHz Intel Xeon, running Linux RedHat, with 8 GB of memory, 6 MB cache, and a 2-disk 1Tbyte striped RAID array. In our experiments we used two widely-accepted benchmarks: LUBM[13] and UOBM [13], for which we considered an instance of 12.8 million triples. Since the expressiveness of the meta-data in the above data sets exceeds the one of Datalog$^{\pm}$ (i.e., OWL-Lite for LUBM and OWL-DL for UOBM), in our experiments we manually produced the FO-reducible approximation of the constraints for all the data sets. These approximations are available on the Nyaya Web site for the reproducibility of the experiments. We have also used Wikipedia3, a conversion of the English Wikipedia[14] into RDF. This is a monthly updated data set containing around 47 million triples. The expressiveness of its meta-data completely falls within that of Datalog$^{\pm}$ so no adaptations were made. The performance of the systems has been measured with respect to: (i) data loading, (ii) querying and reasoning, and (iii) maintenance.

Data loading. As shown in Table 2, Nyaya behaves much better than IODT and BigOWLim. Actually, the main advantage of our approach is that we execute pure data loading while the other systems have to pre-compile the knowledge-base to materialize it, producing a very large (and possibly exponential) number of tuples to be stored. The loading time for BigOWLim degrades dramatically from LUBM to UOBM (135 times slower) due to the increasing complexity of the meta-data, and from LUBM to Wikipedia3 (532 times slower), due to the large set of facts. Moreover our framework exploits the storage meta-model presented in Section 2 which allows a parallel execution of data import into the DBMS. BigOWLim has to process all the triples in memory while IODT needs to maintain complex block indexes. Since for REQUIEM we used our storage model, a comparison with it is not meaningful here.

Querying and reasoning. We performed *cold-cache* experiments (by dropping all file-system caches before restarting the systems and running the queries) and *warm-cache* experiments (without dropping the caches). We repeated all the tests three times and measured the mean execution times. Three phases of the query process have been considered: *preprocessing*, *execution* and *traversal*. In the preprocessing phase BigOWLim loads statistics and indexes in main memory, while Nyaya and REQUIEM compute the rewriting of queries. IODT does not require preprocessing.

We executed the sets of queries included in the LUBM and UOBM benchmarks (they are not reported here due to space limitation). A significant result is the *speed-up* between our approach and the others. We computed the speed-up for all data sets as the ratio between the execution time of each competitor approach P, and that of our approach Nyaya, or briefly as $S = {}^{t_P}/_{t_{\text{Nyaya}}}$. The speed-up for UOBM is reported in Figure 4. The vertical axis is the speed-up S computed for each competitor approach while on the horizontal axis we have the 13 queries. We also report the geometric mean (GM) of speed-up values for both cold-cache and warm-cache experiments. In general BigOWLim performs better than IODT, and Nyaya performs very well with respect to both of them: for cold-cache experiments, the mean Nyaya is 1,6 times and 1,9 times faster than BigOWLim and IODT respectively. For some queries BigOWLim and IODT are faster than our system since the query rewriting produces a large number of CQs to be executed. In these cases we have $S < 1$. Actually, we are currently investigating

[12]`http://www.alphaworks.ibm.com/tech/semanticstk`
[13]`http://swat.cse.lehigh.edu/projects/lubm/`
[14]`http://labs.systemone.net/wikipedia3`

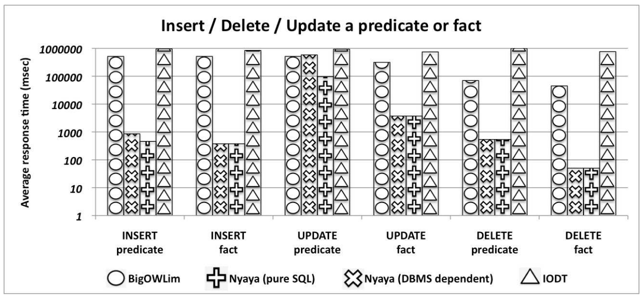


Figure 5: Maintenance of a kiosk

further optimization techniques to significantly reduce this number. Nyaya is 25 times faster than REQUIEM (for the sake of readability in Figure 4 the maximum value on the vertical axis is 10). This is due to the query rewriting phase of REQUIEM, that is the most expensive operation of the overall process. In [10] we reported in detail performance evaluation of each query. We omit the results for LUBM because the systems present a similar behavior.

In the experiments done on the Wikipedia3 data set, Nyaya behaved in general better than the other systems, while IODT performed better than BigOWLim. This is due to the significant amount of data to process, although the set of ontological constraints are simpler in this case. For space limitations, the details have been presented in [10].

Maintenance. In these last experiments we tested maintenance operations in terms of insertion (deletion) of new (old) terms or assertions, and update of existing terms or assertions. Figure 5 shows the performance of each system for update operations of data and meta-data over the LUBM, UOBM and Wikipedia data sets. As for data loading, the comparison with REQUIEM is meaningless. We considered the mean response times (in msec) to insert, delete and update a meta-data predicate (i.e., a concept, a role or an attribute) or a fact in the database. The results show how Nyaya outperforms the other systems when dealing with insertions and deletions of both data and meta-data. In fact, in both BigOWLim and IODT the insertion a new predicate or a fact entails re-compiling a (possibly exponential) number of tuples referring to the new predicate while in Nyaya this simply requires an insertion or a deletion of a single-tuple. On the other hand, the update of an existing predicate needs to be propagated in order to satisfy the constraints. As a consequence, the complexity of this task depends on the amount of data stored in the database. In this respect, Figure 5 shows the behavior of Nyaya in the case of meta-data updates when pure-SQL operations are used (i.e., independently of the DBMS) and when optimized partition-maintenance functions provided by the specific DBMS are adopted.

5. CONCLUSION AND FUTURE WORK

In this paper we presented Nyaya, a system for the uniform management of different repositories of Semantic Web data. This work opens a number of interesting and challenging directions of further research. A natural step is to further extend Nyaya to deal with non-FO-reducible languages like *Guarded* Datalog$^{\pm}$ [5]. Moreover, on a different front, it would be interesting to exploit Nyaya for supporting also the reconciliation and the integration of the data of the Semantic Data Market.

6. REFERENCES

[1] A. Acciarri, D. Calvanese, G. de Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, and R. Rosati. QuOnto: Querying ontologies. In *AAAI*, pages 1670–1671, 2005.

[2] P. Atzeni, P. Cappellari, R. Torlone, P. Bernstein, and G. Gianforme. Model-independent schema translation. *VLDB J.*, 17(6):1347–1370, 2008.

[3] C. Beeri, A. Levy, and M. Rousset. Rewriting queries using views in description logics. In *PODS*, pages 99–108, 1997.

[4] B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, and R. Velkov. OWLIM: A family of scalable semantic repositories. *J. of Web Semantics*, 2(1):33–42, 2011.

[5] A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *PODS*, pages 77–86, 2009.

[6] A. Calì, G. Gottlob, and A. Pieris. Advanced processing for ontological queries. In *VLDB*, pages 554–565, 2010.

[7] D. Calvanese, G. de Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite Family. *J. of Automated Reasoning*, 39(3):385–429, 2007.

[8] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about Datalog (and never dared to ask). *IEEE TKDE*, 1(1):146–166, 1989.

[9] E. Chong, S. Das, G. Eadon, and J. Srinivasan. An efficient SQL-based RDF querying scheme. In *VLDB*, pages 1216–1227, 2005.

[10] R. De Virgilio, G. Orsi, L. Tanca, and R. Torlone. Reasoning over large semantic datasets. Technical Report RT-DIA-149, University Roma Tre, 2009.

[11] G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *ICDE*, 2011.

[12] C. Lutz, D. Toman, and F. Wolter. Conjunctive query answering in the description logic $\mathcal{EL}$ using a relational database system. In *IJCAI*, pages 2070–2075, 2009.

[13] L. Ma, Y. Yang, Z. Qiu, G. T. Xie, Y. Pan, and S. Liu. Towards a complete OWL ontology benchmark. In *ESWC*, pages 125–139, 2006.

[14] H. Pérez-Urbina, B. Motik, and I. Horrocks. Tractable query answering and rewriting under description logic constraints. *J. of Applied Logic*, 8(2):151–232, 2009.

[15] M. Vardi. On the complexity of bounded-variable queries. In *PODS*, pages 266–276, 1995.

[16] R. D. Virgilio, F. Giunchiglia, and L. Tanca, editors. *Semantic Web Information Management - A Model-Based Perspective*. Springer Verlag, 2010.

[17] K. Wang and L. Yuan. First-order logic characterization of program properties. *IEEE TKDE*, 6(4):518–533, 1994.

[18] Z. Wu, G. Eadon, S. Das, E. I. Chong, V. Kolovski, M. Annamalai, and J. Srinivasan. Implementing an inference engine for rdfs/owl constructs and user-defined rules in oracle. In *ICDE*, pages 1239–1248, 2008.

[19] J. Zhou, L. Ma, Q. Liu, L. Zhang, Y. Yu, and Y. Pan. Minerva: A scalable owl ontology storage and inference system. In *ASWC*, pages 429–443, 2006.